WoodyWidgetReference

field widget

The field widget is the most common widget. It is used both for text boxes or selection lists. It can be associated with different datatypes such as string, long or date to ask for different types of data.

[pasted the following text here from the sample, still needs some editing to fit in]

A field widget can be associated with a datatype. The function of the datatype is to convert the string value entered by the user to a more specific type like a number or a date (and vice versa, convert them back to strings) (this part is actually delegated to a seperate object: a Convertor). The datatype also performs the validation. (This split-up between "widget" and "datatype" is specific for the field widget – it is perfectly possible to make widgets that have nothing to do with datatypes). In this way, a field widget contains strongly-typed data. For example, if you associated a **long** datatype with a field widget, then you can be sure that when you retrieve the widget's value, you will get a Long object after the form was validated successfully.

The reasoning behind the **base** attribute on the **wd:datatype** element is that you are actually defining a new datatype, based on the built-in "string" or "long" datatype, and customise it with validation rules (and possibly other parameters).

Defining a selection list for a field

A field widget can furthermore be associated with a selection list. This makes that the field widget could be rendered either as a textbox or a list, depending on whether its datatype has a selection list. For an example of selection lists, see the "Form1" example provided with Woody. The selection-list is related with the datatype: the values in the selection-list should be of the same type as the datatype.

http://outerthought.net/~bruno/images/field_datatype_relation.png

Note: in later versions of woody, and in its successor 'cforms' this set of relations slightly changed, please refer to the cforms documentation for an update

If we wouldn't make these datatype and selection list associations, we would need to create specific widgets for each possible combination: StringField, LongField, DateField, StringSelectionList, LongSelectionList, ...

Configuration example:

```
<wd:field id="..." required="true|false">
  <wd:label>...</wd:label>
  <wd:hint>...</wd:hint>
  <wd:help>...</wd:help>
  <wd:datatype base="...">
     [...]
  </wd:datatype>
  <wd:selection-list .../>
  <wd:validation>
     [...]
  </wd:validation>
  <wd:on-value-changed>
    ...
  </wd:on-value-changed>
</wd:field>
```

The field element takes a required **id** attribute. This id should be unique among all widgets in the same container (usually the form).

The **required** attribute is optional, by default it is `false`. It indicates whether this field is required.

The **wd:label** element contains the label for this widget. This element is optional. It can contain mixed content. For internationalised labels, use i18n-tags in combination with Cocoon's I18NTransformer.

The **wd:hint** element contains a hint for the form control of this widget. This element is optional. It can contain a hint about the input control. For internationalised labels, use i18n-tags in combination with Cocoon's I18NTransformer.

The **wd:help** element contains more help for the form control of this widget. This element is optional. It can contain text help about the input control. For internationalised labels, use i18n-tags in combination with Cocoon's I18NTransformer.

The **wd:datatype** element indicates the datatype for this field. This element is required. The **base attribute** specifies on which built-in type this datatype should be based. The contents of the wd:datatype element can contain further configuration information for the datatype. The possible datatypes and their configuration options are described WoodyDatatypeReference.

The **wd:selection-list** element is used to associate a selection list with this field. See WoodyDatatypeReference for more details.

The **wd:validation** element specifies widget validators. See WoodyValidationRuleReference for more details.

The **wd:on-value-changed** element specifies event handlers to be executed in case the value of this field changes. See also WoodyEventHandling. The interface to be implemented for Java event listeners is `org.apache.cocoon.woody.event.ValueChangedListener`. The WidgetEvent subclass is `org.apache.cocoon.woody.event.ValueChangedEvent`.

Note: *Events used in <`wd:on-value-changed`> require that the form instance is stored serverside (because otherwise Woody doesn't know what the previous values of the fields were). This is automatically the case when you use flowscript. If you don't use flowscript you could store the form instance in e.g. the session.*

multivaluefield widget

The **wd:multivaluefield** is similar to the field widget but can take multiple values. A multivaluefield should always have a **wd:datatype** element combined with a **wd:selection-list**, since the user will have to select values from this list. The wd:multivaluefield could be rendered as a list of checkboxes or as a listbox in which the user can select multiple items.

Configuration example:

```
<wd:multivaluefield id="...">
  <wd:label>...</wd:label>
  <wd:help>...</wd:help>
  <wd:hint>...</wd:hint>
  <wd:datatype base="...">
    [...]
  </wd:datatype>
  <wd:selection-list>
    <wd:item value="...">
      <wd:label>...</wd:label>
    </wd:item>
    [...]
  </wd:selection-list>
</wd:multivaluefield>
```

Most of the elements and attributes have the same meaning as for the field widget.

Note: *A multivaluefield cannot have a `required` attribute, instead you should use the `value-count` validation rule to check the number of values the user has selected.*

booleanfield widget

A **wd:booleanfield** is a field that has a value of true or false. Usually is rendered as a checkbox.

Configuration example:

```
<wd:booleanfield id="...">
  <wd:label>...</wd:label>
  <wd:help>...</wd:help>
  <wd:hint>...</wd:hint>
</wd:booleanfield>
```

repeater widget

A **wd:repeater** widget is a widget that repeats a number of other widgets. It can be used to create e.g. tables, subforms, etc.

Configuration example:

```
<wd:repeater id="..." initial-size="...">
  <wd:widgets>
    [...]
  </wd:widgets>
</wd:repeater>
```

The **wd:widgets** element should contain a number of other widgets to repeat. This can be any of type of widget: field, multivaluefied, booleanfield, or even repeater itself.

The optional **initial-size** attribute allows to specify how much rows should be initially present on the repeater. It mostly avoids to display a table with only table headers. Default value is zero.

Note: *The WoodyTemplateTransformer has specific support for specifying a template to use to render each of the rows of a repeater widget. See the "Form1" sample of Woody for an example on how to use this.*

output widget

An **wd:output** widget is similar to a field widget, but its value is not editable. The value of an output widget must be set programmatically (or through WoodyBinding). An output widget does not read its value from the request, so is most useful in the case where the form is stored accross requests (e.g. as part of a flowscript or flow-apple). An output widget does not perform any validation, it is always considered to be valid.

```
<wd:output id="...">
  <wd:label>...</wd:label>
  <wd:help>...</wd:help>
  <wd:hint>...</wd:hint>
  <wd:datatype base="...">
     [...]
  </wd:datatype>
</wd:output>
```

action widget

Used to trigger an action event on the server side. Usually presented as a button the user can press (though this is not required). When an action widget was activated, validation will not be performed. This is because usually it would be strange to have other fields validated when the user's intention wasn't really to submit the form. If you want validation to happen, use the submit widget. After pressing an action button, the form will normally always be redisplayed, unless the event handling code explicitely disables this (by using the endFormProcessing method on the form object).

```
<wd:action id="..." action-command="...">
  <wd:label>...</wd:label>
  <wd:help>...</wd:help>
  <wd:hint>...</wd:hint>
  <wd:on-action>
    ...
  </wd:on-action>
</wd:action>
```

The **action-command** attribute specifies a name that will be part of the event generated by this widget. It can be used to distinguish events originated from this wd:action from another one.

For more information on how event handlers are defined, see WoodyEventHandling. The interface to be implemented for Java event listeners is `org.apache.cocoon.woody.event.ActionListener`. The WidgetEvent subclass is `org.apache.cocoon.woody.event.ActionEvent`.

submit widget

The submit widget, usually rendered as a button, is used by the user to submit the form. The submit widget is a special kind of action widget, thus also has the same functionality as an action widget, however the submit widget does trigger validation and its purpose is to end the form.

```
<wd:submit id="..." action-command="..." validate="true|false">
  <wd:label>...</wd:label>
  <wd:help>...</wd:help>
  <wd:hint>...</wd:hint>
  <wd:on-action>
    ...
  </wd:on-action>
</wd:submit>
```

The optional attribute validate, which is true by default, can be used to disable validation. The difference between an action widget and a submit widget with validate="false" is that a submit widget with validate="false" will end form processing, thus the form will not be redisplayed (ultimately, it is of course the controller who decides this, but the forms hint towards the controller is that it shouldn't be redisplayed, and this is exactly what the flowscript integration library does).

repeater-action widget

This is a specific type of action widget that handles the much needed case of adding or removing rows from a repeater.

```
<wd:repeater-action id="..." action-command="delete-rows|add-row" repeater="..." select="...">
  <wd:label>...</wd:label>
  <wd:help>...</wd:help>
  <wd:hint>...</wd:hint>
  <wd:on-action>
    ...
  </wd:on-action>
</wd:repeater-action>
```

The **action-command** attribute should have either the value `delete-rows` or `add-row`. If `add-row` is specified, the attribute repeater is required. If `delete-rows` is specified, both the repeater and select attributes are required.

The **repeater** attribute should contain the id of the repeater widget on which this repeater-action should act. This must be a sibling of the repeater-action widget (see also row-action for actions inside a row).

The **select** attribute should contain the id of the booleanfield widget (or any type of widget who's getValue() method returns a boolean) that is part of the repeater and used to mark the rows to be deleted.

wd:on-action allows additional event handlers to be defined, see also WoodyEventHandling. The interface to be implemented for Java event listeners is org.apache.cocoon.woody.event.ActionListener. The WidgetEvent subclass is `org.apache.cocoon.woody.event.ActionEvent`.

row-action widget

This is a specific type of action widget that handles frequent actions occuring on a repeater row, such as adding/removing a row and moving it up and down. These widgets should be placed inside a repeater and act on the current row.

```
<wd:row-action id="..." action-command="add-after|delete|move-up|move-down">
  <wd:label>...</wd:label>
  <wd:help>...</wd:help>
  <wd:hint>...</wd:hint>
  <wd:on-action>
    ...
  </wd:on-action>
</wd:row-action>
```

The **action-command** attribute should have either the value `add-after`, `delete`, `move-up` or `move-down`.

wd:on-action allows additional event handlers to be defined, see also WoodyEventHandling. The interface to be implemented for Java event listeners is org.apache.cocoon.woody.event.ActionListener. The WidgetEvent subclass is `org.apache.cocoon.woody.event.ActionEvent`.

Where all you want to do is submit a specific row on a repeater, simply add a <wd:submit> element to the widgets for the repeater.

Then, you can access the submitted row either using an event handler with `event.getSourceWidget().getParent()`, or from the flow using `form.getWidget().getSubmitWidget().getParent()`. The row itself has a getWidget(widgetName) method that can be used to access specific widgets for the row.

aggregatefield widget

used to edit the value of multiple fields through one textbox or to edit one value through multiple texboxes.

documentation to be done

upload widget

This widget allows to upload files by using Cocoon's file upload features. For this reason, this widget won't function properly unless `enable-uploads` is set to `true` in `WEB-INF/web.xml`.

Also, don't forget to put the **enctype** attribute as `multipart/form-data` in the **wt:form-template** element, inside the template file.

```
<wd:upload id="..." mime-types="text/plain, text/xml" required="true|false">
  <wd:label>...</wd:label>
  <wd:help>...</wd:help>
  <wd:hint>...</wd:hint>
</wd:upload>
```

The optional `mime-types` attribute allows to specify a comma-separated list of mime-types which are accepted. The widget will be invalid if the uploaded type isn't of one of the specified content types.

messages widget

documentation to be done